YOLO

Abstract

- YOLO는 object detection에 쓰이며, 기존의 모델들과는 다르게 regression problem으로 한 번에 end-to-end로 물체 검출
- 입력한 사진 내에서 모델은 bounding boxes를 predict하고 각각의 물체의 class probabilities를 한번의 evaluation으로 결론냄

1. Introduction

DPM

- 이미지 픽셀 (특정 픽셀을 건너뛰고)마다 sliding window 방식으로 box를 그림

→ classifier가 똑같은 크기의 공간을 모두 돌아야 해서 비용이 많이 듦

R-CNN

- sliding window의 문제점을 보완하기 위해, region proposal 방식을 도입
- Region proposal(selective search), Region classification 2중 구조를 띔

→ 각 요소를 개별적으로 학습시켜야해서 최적화가 어렵고 파이프라인 자체가 매우 느려짐

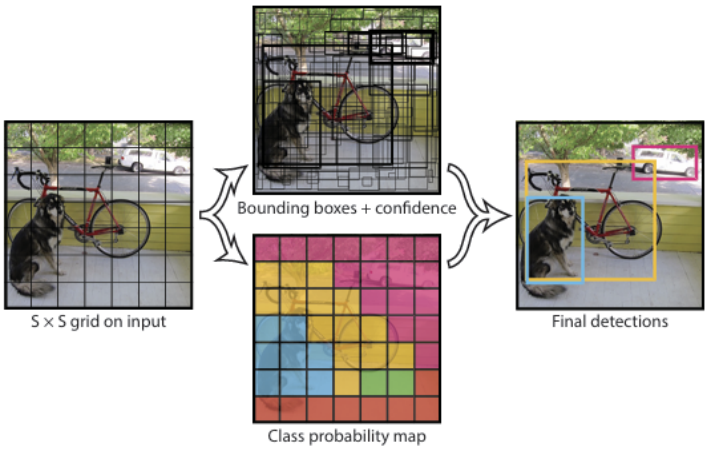
YOLO

- region proposal, feature extraction, classification, box regression 과정들을 one-stage detection에 끝내는 것이 핵심
- 독특한 점은, obejct detection 문제를 classification이 아닌 regression에 해당하는 문제로 치환

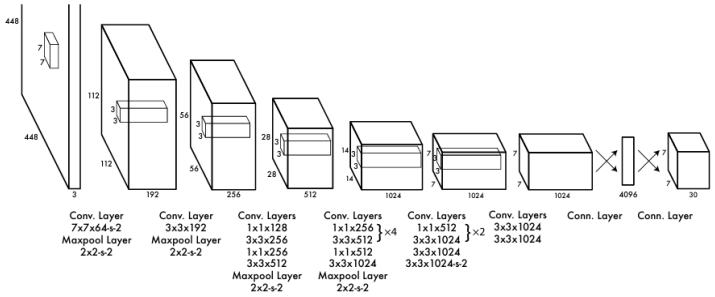
1. 이미지를 448x448로 morph하고
2. 단일 conv net을 돌려 box들과 그에 대한 class probability를 예측 (regression)
3. NMS(Non-Maximum Suppression)를 통해 최종 detection을 수행

2. Unified Detection

- 이미지를 통째로 넣음으로서 global한 특성을 반영하여 detection할 수 있음
- 우선 Input 이미지를 S x S 크기의 Grid로 나누고 각 그리드 안의 물체에 대해 검출을 진행
- 각각의 grid cell은 B개의 bounding boxes와 confidence를 도출
- box안에 물체가 들어 있다는 확신의 정도와 얼마나 정확하다고 생각하는지 나타내는 지표인 Confidence의 값은 Pr(Obj) x IOU



2.1 Network Design



- YOLO 모델은 GoogLeNet Model에서 영감을 받아 24개의 convolutional layers와 2개의 fully connected layers와 1x1 reduction layers와 3x3 convolutional layer를 가짐

2.2 Training

- Layer들에 ReLU대신 Leaky ReLU를 사용

객체가 있으면 1, 객체가 없으면 0

5, 중요도 up

객체가 있을 때 bbox의 위치 (x, y)에 대한 SSE

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

객체가 있을 때 bbox의 크기 (w, h)에 대한 SSE

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

객체가 있을 때 신뢰도에 대한 SSE

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

객체가 없을 때 신뢰도에 대한 SSE

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

0.5, 중요도 down

객체가 있을 때 class probability에 대한 SSE

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

- box의 위치와 관련된 파트는 5를 곱하고, 객체가 없는 상황에 대한 파트는 0.5를 곱해서 더해줌으로써 중요도를 loss에 반영
- 위의 두줄은 모든 grid cell에서 예측한 B개의 bbox좌표와 GT box 좌표의 오차를 구하는 부분
- 다음 두줄은 모든 grid cell에서 예측한 B개의 Pr(Class|Object)와 GT 값의 오차를 구하는 부분
- 마지막 한줄은 모든 grid cell의 Pr(Object)*IOU(truth pred) 예측값과 GT box 값의 오차를 구하는 부분

2.3 Inference

- YOLO 모델은 한번의 계산 이후에 결과가 도출되므로 굉장히 빠름
- 물체가 grid cell 안에 정확히 들어와 있으면 문제가 없는데, 겹칠 경우 검출이 잘 안될 수 있음
- Non-maximal suppression (NMS) 사용하여 위 문제 해결

2.4 Limitations of YOLO

- 하나의 grid cell 안에 bounding box의 수를 2개로 설정했기 때문에 YOLO는 작은 물체를 검출하는데 어려움
- 새로운 환경이나 다른 모습의 같은 물체를 검출할 때 어려움
- 원래 작은 상자일수록 error에 민감해야하나, 현재의 loss function에서는 미미하게 반영되는데 이는 incorrect localization(= bounding box)를 잘못 그리기 때

3. Comparison to Other Detection Systems

Other Fast Detectors

- Fast와 Faster R-CNN은 computation sharing과 Selective Search 대신 neural network를 사용하여 속도를 올리긴 했으나, 실시간으로 사용하기엔 무리

Deep Multibox

하나의 물체를 검출하는데 쓰이므로 YOLO에 비해 부족

OverFeat

- Sliding window 형식으로, convolutional network를 통해 이를 잘 수행
- localization은 잘 해내지만, 전체적인 문맥이 반영되지 못하므로 detection은 잘 못함

MultiGrasp

- YOLO의 grid로 나누는 접근의 배경
- 물체가 들어있을 법한 하나의 region만 도출해내고, bounding box의 크기, 위치, 비율, 그리고 classification은 수행하지 않음

4. Experiments

- YOLO의 경우 background mistake이 적은 대신 mAP값이 좀 작음
- Fast R-CNN은 localization을 잘하는 대신 backgroud error가 많음

- 따라서 이 둘을 ensemble하여 사용하기도 함

https://velog.io/@hewas1230/YOLOv1

https://velog.io/@gr8alex/논문-리뷰-You-Only-Look-OnceUnified-Real-Time-Object-Detection